Multi-User training log for athletes and coaches

Gareth McAteer
BSc Information Systems
Session (2006/2007)

The candidate confirms that the work submitted is their own and the appropriate credit has been given where reference has been made to the work of others.

I understand that failure to attribute material which is obtained from another source may be considered as plagiarism.

(Signature of student) _____

Summary

The aim of this project was to further develop a prototype of a web based training log to support multiple users. The current prototype only supports single users so it was further developed to support multiple users and provide an interface to allow coaches to track an athlete as well offering query and analysis tools. This was done by following a software development methodology to support of the design and implementation of the prototype. Background research into available technologies, software methodologies and evaluation of current web based train logs was also carried out.

Acknowledgements

I would like my supervisor Nick Efford for his help, support and guidance throughout this project.

I would also like to thank my assessor Martin Dyer for his insightful comments and feedback in the mid project report and progress report.

Finally I would like to thank my parents and my sister for helping me get through times of stress during the project.

Contents

Chapter 1: Introduction **1**
1.1 Problem definition 1
1.2 Project Aim 1
1.3 Project Objectives 1
1.4 Minimum Requirements 2
1.5 Deliverables 2
1.6 Project Schedule 2
1.7 Relevance to Degree 3

Chapter 2: Background Research **4**
2.1 Introduction 4
2.2 Running / Training Logs 4
2.2.1 What are training / running logs? 4
2.2.2 Existing Solutions 4
2.2.2.1 Running-Log.com 4
2.2.2.2 RunningAHEAD 5
2.2.3 Summary 5
2.3 Current Prototype 5
2.4 Usability 6
2.4.1 What is Usability? 6
2.4.2 Importance of Usability 6
2.4.2 Usability Checklist 7
2.5 Available Technologies 7
2.5.1 Server Side Technologies 7
2.5.1.1 Django 7
2.5.1.2 Java Server Pages (JSP) 8
2.5.1.3 ASP.NET 9
2.5.1.5 Ruby on Rails 9
2.5.2 Databases 10
2.5.2.1 PostgreSQL 10
2.5.2.2 MySQL 10
2.5.2.3 SQLite 10
2.5.3 Chosen Technologies 10
2.6 Requirements 11
2.6.1 Functional Requirements 11
2.6.2 No Functional Requirements 12

Chapter 3: Methodologies **13**
3.1 Possible Methodologies 13
3.1.1 RAD 13
3.1.2 WISDM 14
3.1.3 Prototyping 15
3.1.4 Agile Unified Process 16
3.2 Chosen Methodology 16

Chapter 4: Inception Phase **18**

4.1 Introduction 18
4.2 Using the Current Prototype 18
4.3 Use Cases 18
4.4 Requirements Analysis 19
4.4.1 Actors 21
4.4.1.1 Actors in the system 21
4.4.2 Organising Use Cases 21
4.4.3 MuSCoW 21
4.4.4 Detailed Use Cases 22
4.5 Prototyping 23

Chapter 5: Elaboration Phase 25
5.1 Introduction 25
5.2 Using Use Case Descriptions 25
5.3 Use Case Realisation 26
5.3.1 Activity Diagram 26
5.4 Design 28
5.4.1 Django and MVC 28
5.4.2 Model Layer 28
5.4.2.1 Database 28
5.4.3 View Layer 30
5.4.4 Template Layer 31
5.4.4.1 Template design 31

Chapter 6: Construction Phase 33
6.1 Introduction 33
6.2 Model Layer and Database 33
6.3 Template Layer and User Interface 34
6.4 View Layer and Business Logic 35
6.4.1 Single User to Multi User System 35
6.4.2 Coach Interface 35
6.4.3 Analysis Tools 36
6.4.4 RSS Feeds 38

Chapter 7: Testing 39
7.1 Introduction 39
7.2 Initial Testing 39
7.3 User Testing 39
7.4 User Interface 40

Chapter 8: Evaluation 41
8.1 Introduction 41
8.2 Methodology 41
8.3 Schedule 41
8.4 Minimum requirements 42
8.5 Technologies 42
8.6 System Evaluation 43
8.6.1 Correctness 43
8.6.2 Robustness 44
8.6.3 Usability 44
8.6.4 Security 45
8.7 Improvements 46
8.8 Conclusion 46

References 47

Appendix A: Personal Reflection **49**

Appendix B: Gnatt Charts **50**

Appendix C: Use Case Description Form Example **52**

Appendix D: Screen Shots **54**

Appendix E: Screen Shots of alternative systems **57**

Chapter 1: Introduction

1.1 Problem definition

Running is popular activity in today's society. A number of people are quite serious about it and invest a lot of time improving their fitness and technique through training. There are a number of factors that can affect a person's performance such as the type of footwear they use, the distance they run, the type of surface they run on and the amount they train. Most that are serious are advised to keep some of log. It helps you have a retrospective view on your training and let you view what works and what doesn't **[1]**.The vast majority would either keep a paper based log or one on a spreadsheet or some other low-tech method. This could be fine for a basic log but many athletes wish also to analyse the data recorded in a log and the previous methods do not offer this functionality. Also athletes may wish to allow their coach or coaching staff to view their training data to help them develop techniques to help improve the performance of the athlete and to help plan future training. Again simple logs as previously mentioned make this difficult.

It therefore believed that a web application would help athletes monitor and record information regarding their training and races or event competed in. It would also allow coaches keep track of an athlete's performance and monitor their progress in an easier manner than is currently available.

1.2 Project Aim

The aim of this project is to further develop a prototype of a web-based training log so that it supports multiple users (athletes and coaches) and allows training data to be analysed in various ways.

1.3 Project Objectives

1. Research existing systems already available.
2. Evaluate current prototype.
3. Research and choose a methodology to follow.
4. Research Django and compare to other web frameworks.
5. Determine a list of requirements for the system.
6. Further develop current prototype with requirements.

7. Evaluate the application in terms of usability and meeting requirements.

1.4 Minimum Requirements

The minimum requirements are:

1. Further develop current prototype to support multiple users.
2. Develop User Interface to support coaches.
3. Create tools for Query and Analysis of training data.

Possible extensions that could be done are:

1. Integrate a mapping tool to plot the areas where athletes have run.
2. Develop a message board for different users to discuss training.
3. Incorporate Ajax techniques into user interface.
4. Produce a user manual to support application.
5. Add RSS feed for training log.

1.5 Deliverables

The deliverables for this project are:

1. A working prototype of the system
2. A Project Report

1.6 Project Schedule

Task	Start Date	End Date
Identify Aims and Minimum Requirements	09-10-2006	20-10-2006
Background Reading	23-10-2006	07-12-2006
Requirements Analysis	23-10-2006	24-11-2006
Rough Prototype	20-11-2006	16-12-2006
Methodologies	30-10-2006	08-12-2006
Mid project Report	20-11-2006	16-12-2006
Christmas Holiday	08-12-2006	20-01-2007
Inception	22-01-2007	31-01-2007
Design Use case 1	01-02-2007	09-02-2007
Design Use Case 2	12-02-2007	20-02-2007
Design Use Case 3	21-02-2007	01-03-2007

Code Use case 1	02-03-2007	22-03-2007
Code Use case 2	22-03-2007	29-03-2007
Code Use case3	30-03-2007	04-04-2007
Progress Meeting	12-03-2007	16-03-2007
Implement addtional Features	10-04-2007	14-04-2007
Testing	05-04-2007	11-04-2007
Evaluation	05-04-2007	21-04-2007
Write report	05-04-2007	25-04-2007

1.7 Relevance to Degree

There are number of modules within Leeds University's School of Computing that were relevant to this project. SE20 and SE24 provided knowledge of how to approach a project like this by teaching skills in analysis and design and the use of methodologies. SY23 covered the skills to design web sites and SY33 provided knowledge of how distributed systems and web applications work and the practices involved in them.

Chapter 2: Background Research

2.1 Introduction

This chapter will report on a background reading required to find a solution to the project problem. It discusses what a training log is and the benefits they bring.. It also researches the different technologies available to build web applications and will go into depth to what Django is and how it is implemented. Also discussed in this chapter are usability and the issues and the requirements

2.2 Running / Training Logs

2.2.1 What are training / running logs?

A training log is a means of recording information about recent training or races the user has recently been involved in. They allow the user to record specifics regarding recent runs they have been involved in such as type of shoe they wore, what surface they ran on, the time they finished their run. It can be also be a useful source of information regarding periods of good training and/or injury **[1]**. They can also be use to give information on how often you need to change your running shoes and signs of over training. They allow also for coaches to view this information and able to base on the recorded information make suggestions to the athlete.

A web based log would allow a coach to manage many athletes concurrently and from many locations meaning the coach and athlete would not have to be immediate contact with each other to share information. This would allow a coach to be actively involved in a number of athletes training regimes and offer advice despite not being in direct contact.

2.2.2 Existing Solutions

There is a number of training log systems already available for use. They support a variety of features, but most offer the same functionality. They allow the athlete to have private area to log information, but the type of data that can be entered differs.

2.2.2.1 Running-Log.com

This solution offers the user quite a basic system. You need to register before using the web application and once you log in you are presented with a number of options. You can enter your workout, view your monthly workout, you can edit or add information regarding shoes and the routes you have run. It also has a reports feature that let's the user view a monthly report or yearly report of the training the user has done. It also offers a report on races you have participated in as well some statistics giving projections of what you average running time is amongst others details **[4]**. This solution offers no interface for coaches.

The web application is written in ASP.net with Microsoft SQL Server as its database.

2.2.2.2 RunningAHEAD

This solution offers a lot more functionality. When you log in, you are presented with a menu across the top allowing you add a new race event, a new shoe or a new course. The left hand menu allows you to view a summary of entered information, a calendar of entered training details and events participated in, a list of entered shoes and courses, personal records, a search function and an option to view a graphical output of entered information. When you are entering new details there is quite a lot data to be entered, some of which is not really necessary, such as purchase date of shoes. The use of graphs to display running distances and times is however useful in showing the athlete patterns of performance **[5]**. This solution also does not offer any interface for coaches.

This solution was also written in ASP.NET.

2.2.3 Summary

Although these two solutions do provide an athlete with a means of keeping a training log, they lack certain functionality. The main thing they lack is ability to allow coaches to play a part in the training log. They offer no options at all for coaches. A coach cannot register on these systems and the athletes who use these training logs have no means to add a coach or anyone for that matter to view the recorded training information. A lack of this feature in these systems means that the run log being developed will have to include this feature.

2.3 Current Prototype

The current Run Log web application developed by Nick Efford is written in python using the Django web application framework and a PostgreSQL database to store data. In its current stage it is very

much a single user application written entirely with only one athlete in mind. There current system has no ability to handle multiple athletes and allow each individual athlete to record personalised information, there is only functionality in place tailored for one individual. The current system also does not have any security mechanisms in place to ensure that only the athlete, who recorded the information, can view, edit and add data. Currently when the application is run anyone can access the athletes training information. There is also no coach interface implemented in the prototype that would allow them to view athletes training logs. There is some statistical functionality currently implemented but there is no graphical output generated and more query tools need to be developed.

Currently the system is a single user system with no functionality in place to support multiple users and a coach interface is non existent, with minimal statistical and query tools present.

2.4 Usability

Usability factors are an import issue when it comes deigning a web application. Jakob Nielsen **[2]** who is a usability guru states: “Usability is a quality attribute that assesses how easy user interfaces are to use. The word "usability" also refers to methods for improving ease-of-use during the design process”. Usability is important issue that cannot be ignored because if a user cannot effectively operate the web application easily they will not continue to use it.

2.4.1 What is Usability?

According to Jakob Nielson usability can be defined five quality components**[2]**: Learnability, Efficiency, Memorability, Errors and Satisfaction. Learnability is how easy it is to complete functions the web application offers when the user first encounters the system. Efficiency deals with once the user has learned the design of the web application how quickly can they perform tasks. The Memorability component deals with if the user has not used the web application for a certain period of time, how easily can they use the system again efficiently. Errors are where if the user makes a mistake how easily can it be corrected and what actual errors does the user make. Finally Satisfaction is how satisfied the user feels when using the web application.

2.4.2 Importance of Usability

Usability is very import factor when it comes to using a web application, if a user finds a web application hard to use, they will tend not to use it, if the web application does not clearly state what it does, people will not use it, if the interface means users get lost, they will not use it[2]. What this is saying is that it is essential to get the usability of the web application to a standard that will not put of

initial users off from using it. Users will not want to read through a web application manual or spend to time trying to figure out the interface, they will simply leave and use another web application [2].

2.4.2 Usability Checklist

In order to make sure this system is as usable as possible a usability checklist will be used. This will aid in the design of the web front end of the system and ensure that the system is as usable as possible. Larisa Thomason **[22]** provides such a check list and in this list the main points are: design a clear and usable web site, design a clear and simple navigation system, keep content clear and simple, support your brand, and test the site on real users. By following this when designing the web interface to application should ensure that usability issues will be reduced.

2.5 Available Technologies

2.5.1 Server Side Technologies

2.5.1.1 Django

Django is web development framework for Python. Python is a high level interpreted object orientated language created by Guido van Rossum in 1990. Some of the features of Python are that is dynamically typed meaning that data types for variables do not have to be declared, Python syntax is also described as concise meaning it takes less code to perform the same task in another language. An example of this is that you can write one line of code in Python that would be equal to 10 lines of code in Java **[10]**.

To develop a web application in Django, code is written in Python that uses Django libraries. Django follows the Model-View-Controller (MVC) architecture. This desirable for a number or reasons, with MVC there is a separation of code and presentation so that presentation is dealt with by templates and the application code is dealt with by Python modules. The MVC approach also encourages a single, abstracted data-access layer that responsible for all data access **[10]**. This means is that the data-access layer knows the database table and allows you execute SQL queries via Python instead of manually writing SQL queries **[10]**.

When a Django application is written there are four areas of the framework to consider. The first part is the 'model'. Django uses an object-relational mapper which means to you model the database layout in python code, this is the model**[11]**. You write a class in the model to describe how your database is going to look; each class represents a new table in the database. The next area is the

'view'. A view is primarily responsible for two things, getting the content for the requested page, through and HttpResponse object and raising exceptions**[11]**. This is not all a view does, it is the part of the framework where the business logic for the web application is written. In a very general sense the view retrieves the data according to parameter, loads the relevant template and renders the template with the appropriate data. Another area of the framework is the URLs. Django encourages URL design that doesn't include the extension of the file like ASP or PHP would. To design the URL, a python module Called URLconf is created. This is essentially a table of the contents for the application and provides simple mapping between URL patterns and python call back functions **[11]**. The last area of the framework to be discussed is the template system. A base template is written in HTML which dictates the general look of your web application's user interface. It also provides 'holes' for the child templates to fill **[11]**, such a news content. This is the idea of template inheritance. When you write a 'child' template you insert code that tells it to extend the base template. Variables can be placed in the template to display content from these variables you look up an attribute of the variable as well as diction-key lookup, index lookup and function calls **[11]**. Within the template you can also for loops and if statements. Django also uses a template search path to try and minimize redundancy among templates, in the setting file you can tell Django in which directories to look for a template, if a template doesn't exist in the first directory it will search the second one and so on **[11]**.

When building an application Django an admin interface is automatically created for the developer so they can concentrate on implementing the functionality for their application with having to also thinking about building administrative tools.

2.5.1.2 Java Server Pages (JSP)

JSP is a server side technology that allows developers to create and maintain dynamic web pages. Because JSP is part of the Java family it enables the developer to develop web based applications that are platform independent. JSP also separates the user interface from the development side, meaning that the developer can change the design of their web page without altering the content generated **[6]**. A JSP page in its simplest form is an HTML web page that also has additional code that is used to generate dynamic content. The full range of Java's API's can be used with JSP such as JDBC (Java Database Connectivity), JavaBeans, and RMI (Remote Method Invocation) **[6]**.

JSP is an extension of the Servlet API. They differ from Java Servlets because they are compiled into Servlets before they are used, meaning they have all the benefits of Servlets, such access to Java APIs **[6]** but they offer a more convenient approach to developing web applications because the code is embedded in the HTML meaning the developer can change the application layer without having to

think about the presentation layer. This is generally a bad idea. JSPs work well provided that the Java code embedded within them is presentation logic rather than business logic. Most Java solutions today use a layer sitting on top of JSPs that help to keep everything cleanly separated, typically following a Model-View-controller architecture.

2.5.1.3 ASP.NET

ASP.NET is a server side technology that uses the Microsoft .NET framework and runs on a web server. A number of different programming languages can be used to develop an ASP.NET web application such as C Sharp (C#) and Visual Basic.NET **[7]**. Unlike Java and its surrounding technologies, ASP.NET is proprietary, but an open-source implementation of the technology is available as part of the Mono platform, which runs on both Windows and Linux **[7]**. ASP.NET makes use of the .NET framework meaning that it collects all the technologies needed for building web applications as well as desktop applications and web services and bundles them into one package. Like JSP, ASP.NET allows the developer to separate the server side code from the presentation layer, meaning that developers can change code without effecting how the web application will look.

When a developer uses ASP.NET to connect to a data source, it uses technology called Active X Data Objects (ADO.NET). The source of the data can be any Relational Database such MS SQL Server, Oracle or MySQL, and other non relational sources such as XML and text files **[8]**. Essentially this means that you can connect to any data source that will run on the windows platform giving you scope to use a variety of database technologies.

2.5.1.5 Ruby on Rails

Ruby is cross platform interpreted object orientated language created by Yukihiro Matsumoto in 1995. It is based heavily on the object orientated language Smalltalk and as such is purely objected orientated, meaning everything is handled as an object.

Rails is web application framework for Ruby commonly known as 'Ruby on Rails' that includes everything required to build database backed web applications using the Model-View-Control pattern of separation. Model-View-Controller (MVC) is a tried and tested design approach, a Model represents an object (such as Person), a View is an output from the model and a Controller essentially links the two together. In Rails, the Model and Controller are Ruby Classes and the Views are XHTML **[9]**. Rails also has object modelling that links to tables in a database meaning you can manipulate table contents without writing SQL.

2.5.2 Databases

2.5.2.1 PostgreSQL

PostgreSQL is an open source, platform independent relational database. It can be run on all major operating systems including Linux, Windows and Mac OS X. It is fully ACID (Atomicity, Consistency, Isolation, and Durability) compliant and has support for features such as foreign keys, triggers, joins, views and stored procedures **[12]**. It also has support for many programming interfaces like C/C++, Java, Python, Ruby and Perl.

2.5.2.2 MySQL

MySQL is also an open source platform independent relational database. It can be used on a variety of operating systems like Linux, Windows and MAC OS X. Like PostgreSQL it is ACID compliant and supports features like foreign keys and joins. MySQL is a client/server system that supports different back ends, different client programs and libraries and a wide range of application programming interfaces **[13]**. It also has a number of security features like a privilege and password system that allows host based verification.

2.5.2.3 SQLite

SQLite is a small C-Library that implements a self-contained embeddable, zero configuration SQL database engine **[14]**. It is ACID compliant and supports features like joins and foreign keys although foreign key constraints are not fully implemented and complete databases can be stored on a single disk file. It is generally used for small scale websites that require a simple database back end.

2.5.3 Chosen Technologies

Because the current prototype that is being further developed and has been significantly developed in Django already it makes sense to continue developing it in Django. This means that application will not have to be re-written from scratch in another language and the current functionality can be used and adapted. Django also has a lot of built in features that will prove helpful in the development process such session management and security features and the pre-built admin application. Also Ihaving had previous experience programming in python will make the development process quicker because I will not have to learn a new language.

The database chosen was SQLite because offers all the functionality that is needed to develop this application. Some of the more in depth functionality offered by MySQL and PostgreSQL are not required for an application of this type. Because SQLite databases are single files it means easy transfer of data from my home computer to the university workststations.

2.6 Requirements

Having discussed the current prototype and the limitations it currently has, it is essential capture the requirements the extended system must have.

2.6.1 Functional Requirements

Athletes

With the current system being a single user system, it is desirable to support the ability for multiple athletes to use the system. Athletes should be able to record information relating only to them and any data they recorded should only be accessible by them. An example being that if an athlete records a new type of shoe, only that athlete must be able to view that show, edit it and use that shoe in other areas of the system. No other athlete must be allowed to access another athlete's recorded data. They must also have to ability to select a coach to view their details. When they register they should be able to choose to be an athlete. An RSS feed of training techniques and running information to allow them to keep up with the last information. A integrated mapping tool allow the athlete to visually plot the routes they use.

Coaches

The current system does not support coaches. There are a number a activities that a coach must be able to perform. A coach must be able to view an athlete's recorded information but must not be allowed to add to it and edit it. The coach must be able to select which athlete to view and be able to switch between athletes at any time. When they register they should be able to choose to be a coach.

Administrator

In order to manage the system there must be some administrator role. When a user registers on the system, they registration must be approved by the administrator before they can use the system. The administrator will not actually be able to use the system, but will have access to everyone's details and be able to edit them. Including training data recorded by the athletes.

Analysis tools

Currently the system provides a few basic analysis tools, but none of the tools provide any graphical output to enable users to analyse data more easily. So athletes and coaches must be provided with more detailed analysis tools giving graphical output. This must be accessible by athletes and to coaches

2.6.2 No Functional Requirements

Database

A database will be required to store information recorded by the user of the system be that the athlete or the coach. The database used

User Interface

A user interface is required so the user can access the system. This will have to be consistent and take into account usability factors to ensure the system is easy to use and intuitive for the user.

Chapter 3: Methodologies

3.1 Possible Methodologies

There are number of different software development methodologies available. A methodology consists of an approach to software development incorporating a set of techniques and notations such as UML to support the approach, a life cycle model to help structure the process of development and unifying set of procedures and philosophy [15].

Before the idea of software methodologies was introduced the development of software was often unstructured leading to systems that were not well designed and often did not meet the requirements of users. Estimating project deadlines was difficult and because of this development overran and without a clear structure to follow, projects overall often overran. Since the introduction and use established methodologies projects are better managed. Although certain problems still exist, methodologies reduce these problems greatly and give projects a structure to follow meaning systems can be delivered more effectively and with fewer delays.

It is therefore essential that a software methodology is followed in this project to ensure that the system is designed properly and meets deadlines. This sections reviews possible methodologies that could be followed.

3.1.1 RAD

Rapid Application (RAD) is software methodology that centres on building systems and applications in a short amount of time. This often comes with compromises in areas such as functionality, features or even execution speed **[16]**. The idea of this framework is to develop and design systems within 60-90 days but it was originally developed to describe the development process that involves application prototyping and iterative design.

The methodology works by dividing development up into different sections called ‘time boxes’. These time boxes state a start and finish time for development of each part of the system. In each of the time boxes functionality is developed as quickly as possible and once the end of the time box is reached development of this part of the system is stopped. The time allowed to each of the time boxes

is strict because any deviation with lengthen the development iteration and limiting feedback from users**[16]**. In the first time box, high level user requirements are implemented meaning that all the main functionality of the system is in place. Subsequent time boxes implement lower level requirements.

Prototyping is another key feature of this methodology; the aim is to build a lightweight prototype as quickly as possible which helps to flush out user requirements **[16]**. The idea of this is provide a proof of concept and provide a talking point and a tool for refining user requirements.

There advantages and disadvantages to using RAD. One of the key advantages is the increased development speed and decreased time to delivery. Delivering applications quickly is achieved by the use of CASE (Computer Aided Software Engineering) tools with the idea being that the requirements are coded as quickly as possible, as well as the use of time boxing which pushes out to future releases in order to complete development as quickly as possible **[16]**. A disadvantage is that often systems can have reduced features, because due to time boxing some features are pushed to later stage in development in order to get a working system in place as quickly as possible **[16]**.

Using RAD allows the changes in user requirements, because the system has not been fully implemented yet. A new requirement can be added to later time box. The use of time boxes prevents projects from overrunning because they cannot be extended but because of this, this can lead to loss of functionality. Using RAD, system can be delivered quickly, on time and with most of the user requirements met, but also can often be found missing some functionality.

3.1.2 WISDM

Web IS Development Methodology (WISDM) is an adaptation of the Multiview Methodology for web development. WISDM provides a framework for using traditional software methodologies combed with web based techniques to provide an overall rounded framework, from analysis to implementation **[18]**. It is a used as methodology for web based systems.

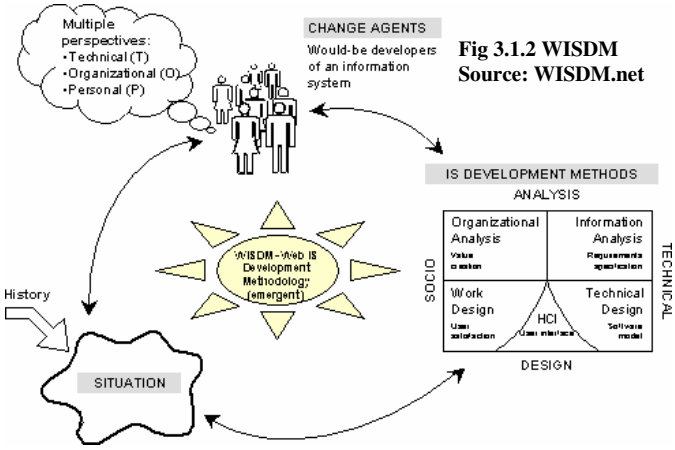


Fig 3.1.2 WISDM
Source: WISDM.net

WISDM makes user of the ISD matrix which is categorised into two dimensions: socio (organisations and

people) and technical, on the one hand analysis and the other design. Engineering approaches to systems development focuses on the right hand of the matrix, the specification and refines this into a software model. The Multiview approach is even, overall focusing on both sides of the matrix gin order produce a more sociotechnical solution **[19]**.

3.1.3 Prototyping

Prototyping is the idea of building version of the final system. There are two kinds of prototyping methodologies: throwaway prototyping and evolutionary prototyping. Throwaway prototyping is where a prototype is built as quickly as possible, often showing how the main functionality will work but in a limited capacity incorporating a basic user interface. From this the users can re-examine their requirements. After this the prototype is 'thrown away', the system is then built again from scratch according to the new requirements established. Evolutionary Prototyping works by building a prototype based on user requirements, and then when the user tests the prototype, any new requirements are built into the prototype. This is process is repeated over a number of iterations until a working live system has been produced.

Prototyping is more a user centred approach to development, more so than RAD or WISDM. No clear user requirements are established; instead the user is presented with light version of the system on which they comment on. This hands on approach by users provides experience and allows them to have an active input in the design of the system **[20]**. This input is either incorporated in the solution, which is evolutionary prototyping or discarded and used to derive user requirements, which is throwaway prototyping.

Prototyping has many benefits. As discussed above it heavily involves the user. This can be used to discuss features that maybe the developer did not consider of but the user has been thinking of. This can lead to overall user satisfaction because the user has been involved and has told the developer the features they expect or want to see in the system **[20]**. It can also offer the developer and user the chance to discuss the feasibility of some of the functionality they desire to have and the technological limitations that could lead to certain desired functionality not being implemented and therefore reducing disappointment when a final solution is produced that differs from user expectations.

There are however some drawbacks to prototyping. Prototyping has been criticised for leading to inadequate or partial system designs. It is argued that designs emerging from prototyping are not properly engineered and that they just emerge **[20].** Prototyping is also seen to be hard to manage due to the fact that there could be an endless cycle of constant small changes leading to projects that have no set end point**.** Many critics also see prototyping as no substitute as a substitute for detailed analysis

and design that happens in other methodologies**[20]**. The requirements that may emerge from the user prototyping may not be useful to the core system.

3.1.4 Agile Unified Process

.The Agile Unified Process (AUP) is a simplified version of the Rational Unified Process (RUP) **[17]**. It is a simple, easy to understand approach to developing software using agile techniques while still upholding the fundamentals of the RUP.

In the lifecycle of the AUP there are four stages: Inception where the aim is develop the initial scope of the project and potential architecture; Elaboration which is proving the architecture of the system; Construction which is building the a working solution in an incremental way which meets the highest priority needs of the project; Transition which is deploying the system **[17]**.

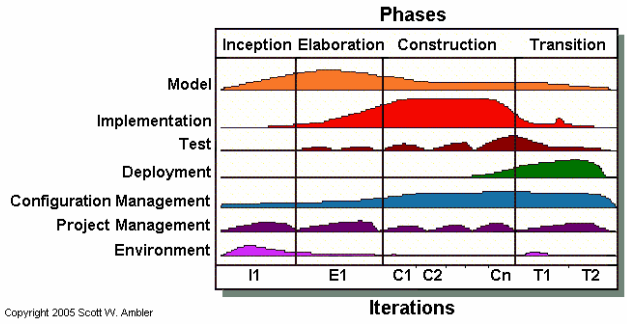


Fig 3.1.4 Phases in AUP
Source: Agile Unified Process

The idea is to produce iterations of the delivered software rather than releasing it all at once, releases of the software are typically made at the end of each iteration. The first release generally is a basic working system with basic architecture in place and subsequent releases after this develop further the initial system with more features added after each release **[17]**. Using this method allows the developer to avoid problems and overcome problems early on and take advantage of experiences during development.

There are also certain philosophies that the AUP stresses. In the AUP developer knows what going on at all times, simplicity is also stressed meaning everything is described in a concise manner. Agility is also stressed meaning things change can be incorporated and the AUP takes this into account. You are not tied to any tool in the AUP either, use the best tool for the purpose whatever it is.

3.2 Chosen Methodology

The methodology that was chosen to follow was the Agile Unified Process. Prototyping was also considered because of the ability to produce solutions very quickly but eventually was rejected due to the fact that there is no clear schedule or process to follow meaning perhaps that project could get out

of control. WISDM was also consider due to the fact that it is a framework designed for use on Web based applications but again was rejected because it may not manage a project of this type adequately.

AUP was chosen because it offers a framework that employs careful management of the project and clearly set out stages to follow. Because it is iterative and different functionality is developed in these iterations it also breaks the overall problem down into more manageable goals rather than presenting the developer with a mammoth task of developing an entire system at once. Because this project is being developed by one individual, the AUP will be modified slightly. Also because of the way the Django works in the fact you don't develop in a true object orientated manner, the AUP will again have to be modified slightly.

Chapter 4: Inception Phase

4.1 Introduction

The first phase of the AUP is the inception phase which aims to define the initial scope of the project and understand the architecture of the system **[17]**. By Understanding the architecture, risks that might arise in the project and the feasibility of the system can be determined. This phase of the AUP focuses heavily on analysis of the problem which will lead on to the design of the system which dealt with mostly in the next phase.

4.2 Using the Current Prototype

By analysing the current prototype it was decided at this phase that to deliver a system that meets the requirements that the current functionality of the system would have to be rebuilt in order for the system to function. Simply plugging in existing functionality would not work and would produce a 'hacked together' solution. By taking into account what was already in the prototype would help in the analysis and design of the system and provide some groundwork from which to start the analysis from.

4.3 Use Cases

Because AUP is a use case driven approach, the requirements of the system are captured as use cases. A use case specifies the scope of the systems and allows the developer understand what the user requires **[15]**. An example use case from the user requirements is that an athlete should be able to set a coach which is shown in figure 4.2. Here it is shown how it would be modelling using UML Use cases are useful in detailing each use case using UML diagrams **[15]**.

Fig 4.1 Set Coach Use Case

4.4 Requirements Analysis

From the requirements in Chapter 2 a list of uses cases was produced by analysing the requirements, which is shown below in table 4.2. Since the AUP is use case driven which means that all the requirements captured during this phase will be used throughout development process. They will be used in analysis and design as well as testing.

Table 4.4 Table of Use Cases

Requirement	Use Case(s)
To store an athlete's own shoe	Add Shoe
To edit the details of an athlete's own shoe	Edit Shoe
To list all the athlete's own shoes	View Shoes
To view how much an athlete's own shoe has been used	Shoe Usage
To store an athlete's own route	Add Route
To edit the details of an athlete's own route	Edit Routes
To list all the athlete's own routes	View Routes
To view how much an athlete's own route has been used	View Routes Usage
To store an athlete's own session	Add Route
To View an athletes own monthly sessions	View Monthly Session
To view athlete's own current weekly sessions	View Sessions
To view an athlete's own distance covered in training	View Distance Covered
To store an athlete's own race	Add Race
To edit the details of an athlete's own race	Edit Race
To list all the athlete's own races	View Distance Covered
To view how much an athlete's own race has been used	Race Usage
To store an athlete's own event	Add Event
To view details of an athlete's own event	Event details
To list all the athlete's own current weekly events	View Weekly Events
To list all the athlete's own current yearly events	View Yearly Events
To view an athlete's own Current coach	View Current Coach
To set an athlete's coach	Set Coach
To view a coach's currently associated athletes	View Athletes

To register a user	Register User
To Log a user in	Log in User
Activating a user on the system	Activate User
To allow coach to view a selected athlete's shoes	View Athlete Shoe
To allow coach to view a selected athlete's shoe usage	View Athlete Shoe Usage
To allow coach to view a selected athlete's routes	View Athlete Route
To allow coach to view a selected athlete's routes usage	View Athlete Route Usage
To allow coach to view a selected athlete's monthly sessions	View Athlete Monthly Session
To allow coach to view a selected athlete's current weekly sessions	View Athlete Sessions
To allow coach to view a selected athlete's current distance covered in training	View Athlete Distance Covered
To allow coach to view a selected athlete's races	View Athlete Races
To allow coach to view a selected athlete's race usage	View Athlete Race Usage
To allow coach to view a selected athlete's event details	View Athlete Event Datails
To allow coach to view a selected athlete's current weekly events	View Athlete Current Weekely Events
To allow coach to view a selected athlete's current yearly events	View Athlete Current Yearly Events
To allow coach to view a selected athlete's yearly best performance	View Athlete Yearly Best Performance
To View an athlete's own RSS feed	View RSS Feed
To add a new athlete's own RSS Feed	Add RSS Feed
To View map of athlete's route	View Map Route
To add to athele's own map of route	Add Map Route
To Register an athlete	Register Athlete
To Register a Coach	Register Coach
To Log in to system	Log In System

The athlete uses cases listed here already exist in the current prototype. But in the current prototype they are only catered for a single user. It was there deemed necessary to model again them to fit in with the ability of the system to handle multiple users that the requirements in chapter 2 set out.

4.4.1 Actors

It is important to identify the actors with in the system. An actor represents the roles of people or anything else such as other systems that interact with the use cases of the proposed system **[15].**

4.4.1.1 Actors in the system

Athlete: The athlete is person who takes part in any events, trains and uses the equipment (shoes) and records the information. This includes adding any new shoes, race, event etc, editing shoes races etc

Coach: The coach is responsible for training and monitoring the athlete. This includes view the information the athlete has recorded such as training sessions, shoes they have, and races they have completed etc

Administrator: The administrator is the manager of the system who is responable for ensure the system is working and managing the data stored by the system. This includes managing accounts and making any changes to user details if needed.

4.4.2 Organising Use Cases

To aid the development process the use cases derived form the requirements were placed into groups that was relevant. The groups that the use cases were placed in were Account management, Athlete Management, Coach Interface, Query Tools, RSS management, Mapping Management. By grouping use cases together like this means the task of developing the different aspects of the system becomes a lot easier and more manageable and since a lot of the requirements in n group are part of the same overall task it makes sense to group them. Athlete Management, Coach Interface, Query Tools, RSS management, Mapping Management all encompass was makes a running log.

4.4.3 MuSCoW

By applying MuSCoW rules, this allows a critical analysis of the requirements and subsequent use cases produced from analysis allowing prioritisation of them **[20].** MuSCoW places the use cases into “Must have”, “Should have”, “Could have” and “Won’t have” categories

Must have: These are features that are essential for the system to have in order to meet it’s purpose and function correctly

Should have: These are features that the system should in but are not essential for the system to meet's its purpose.
Could Have: These a features that could be implemented but are not essential for the system to meet's its purpose.
Won't Have: These are features of the system that will not be implemented.

On looking what the system needed to deliver functionality wise based on the requirements gathered, the system must be a multi –user to enable multiple athletes to use the system effectively, so Athlete management was considered to be an import functionality, the same with the coach interface to allow coaches to view their athletes and monitor them effectively. Query tools were also deemed important but the system could function without them. RSS Feeds and the mapping tool were deemed less important and do not affect how the ability of the system to function, they could be considered 'luxury items'.

Table 4.4.3 MuSCoW Use Cases

Priority	Use Cases
Must Have	Account management, Athlete Management, Coach Interface
Should have	Query Tools
Could Have	RSS management, Mapping Management
Won't Have	None

4.4.4 Detailed Use Cases

T o explain the functionality in more detail, each use case's functionality was briefly described, this allowed the development of each of each use case to be clearer and be less ambiguous.
Below is a table that shows the descriptions for some of the use cases in the Athlete management group.

Table 4.4.4 Use case description example

Use Case	Description
Add Shoe	When a new shoe is described the system adds this shoe. This includes shoe name, type of shoe and when it was first used. The systems stores this shoe against the athlete that entered the data.
View Shoe	Details of the shoes can be viewed. The athlete selects to to view shoes and the system retrieves the shoes for that athlete
Edit Shoe	The shoe stored can be edited by the athlete whose shoe it is. The system makes any changes the athlete has performed

View Shoe Usage	Displays how much a shoe has been used by an athlete. The athlete chooses to view shoe usage and the system retieves the usage details for all the athletes shoes.
Add Event	When is a new event is described the system adds this event. This includes date of the event, the time it starts at, the event race, the duration of the race, position in the event, shoe worn, and any notes they may have. The system stores this event against the athlete that entered the data
Event details	Details of the event can be viewed. The athlete selects to view an events and the system retrieves the events for that athlete
View Weekly Events	Details of the weekly events can be viewed. The athlete selects to view weekly events and the system retrieves the weekly events for that athlete
View Yearly Events	Details of the yearly events can be viewed. The athlete selects to view yearly events and the system retrieves the yearly events for that athlete
View Best Performance	Details of the best performance can be viewed. The athlete selects to view best performance and the system retrieves the best performance for that athlete

4.5 Prototyping

A basic prototype of the system was done in part to prove that what was set out in the uses cases could be accomplished, this included making a prototype of a system that registered users,. It was simple application in Django that took a user's details and stored them in the database. This was useful in the fact it showed that what I wanted to achieve could be done. A simple system that stored a shoe with the athlete hard coded based on the original prototype was also constructed to get an idea of the task involved of creating a multi user system.

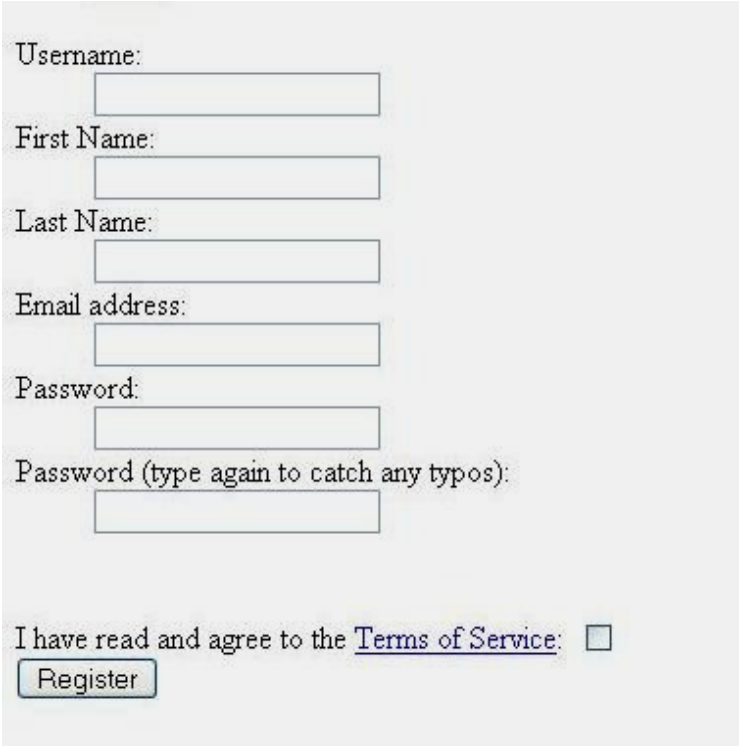


Figure 4.5 Prototype Registration

Chapter 5: Elaboration Phase

5.1 Introduction

The next phase of the AUP is the elaboration phase. In the elaboration phase the goal is to prove the architecture of the system being developed **[17]**. From the analysis in the performed in the inception phase, this phase deals more with the design of the system by developing architecture from the analysis done.

5.2 Using Use Case Descriptions

A use case description allows a use case to be fully described in order to illustrate a use case in more detail. In the inception phase, the use cases of the system were discussed and briefly described in order to get an understanding of them. From this use case description forms are used to show the basic flow, alternative paths or exception paths **[15]**.

The basic flows of events illustrates the general path a user of the system will take that will allow them to achieve a successfully what they are trying to achieve **[21]**. In layman's terms this is the path you generally expect the user to take. The alternative flow of events shows the different path a user may take in order to achieve the same successful outcome. This means that this is the path you don't expect a user to take but may do, in order to achieve a successful outcome. Finally the exception flow path is one that that the user take but will lead to an outcome of no value. By taking this path, the user will not achieve a successful outcome.

Considering all the paths that a user may take is useful because it takes into account how a user may use the system allowing the possibility of errors to be found as well as ensuring that requirements are met. Use case description forms are also useful in the fact that they capture any pre-conditions and post-conditions for that particular use case. Other areas captured in the form include the non functional requirements of the use case. Use case description forms were not used for all of the use cases identified, only a few, due to the fact that a lot of the use cases are similar.

5.3 Use Case Realisation

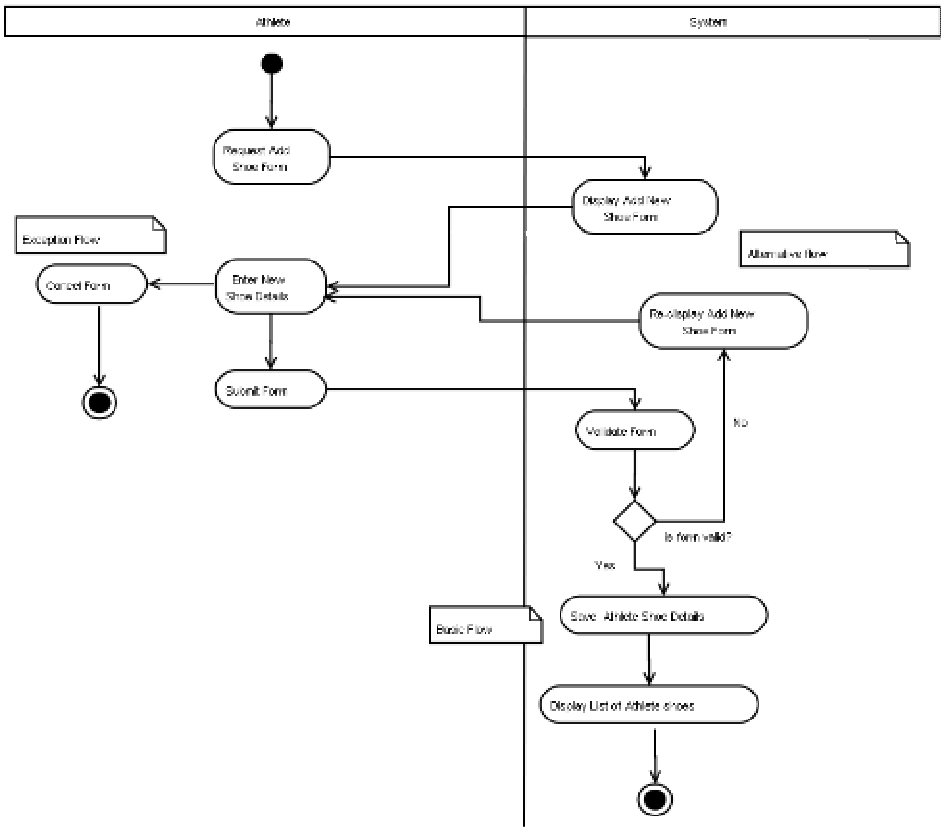
The next stage of this phase is to realise the use cases. This process involves the identification of possible classes allowing an understanding how these classes may interact and allow the functionality of the use case to be delivered **[15]**. A major part of this phase is architectural modelling **[17]** where the purpose is understood how the system will be built or in this case modified. Use case realisation is technique that allows this happen. The use of an activity diagram and a collaboration diagram incorporating analysis classes were used to do this

To show this process the use case for 'add new shoe' will be used as an example.

5.3.1 Activity Diagram

The first stage is draw and activity diagram to visualise the flow of events for this use case. Figure 5.3.2 shows the activity diagram with swim lanes of how the athlete performs the task and how the system is also involved in completing the task. Doing this allowed the identification of the flows of events that could take place. The following activity diagram was derived from the use case form seen in appendix C.

Fig 5.3.2 Activity Diagram



5.3.2 Collaboration Diagram

The next activity to do is to construct a collaboration diagram. This is a lower level view of the use case. In this diagram analysis class are used their stereotypes are used to show the represent the three layers of the 3-tier architecture. There are 3 types of analysis class: Boundary, entity and control. Boundary classes represent the interaction between the system and the actor and can be mapped to presentation layer. Entity classes represent important concepts within the system and information that will be used over a period of time and can be mapped to data layer. Finally, control classes represent the events that will take place within the system and can be mapped to business logic layer [15].

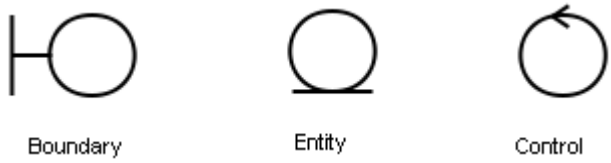
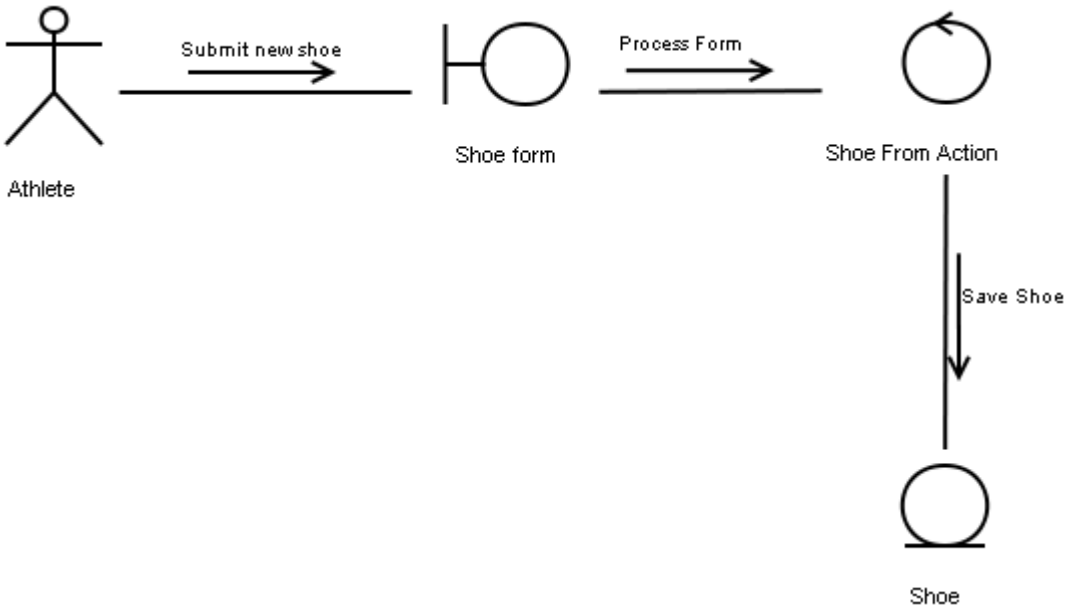


fig 5.3.2a Analysis classes

With an understanding of the use case and how the use case will flow with the use of the activity diagram, this allowed the analysis classes to be considered. From the collaboration boundary class will add the new show form which the athlete uses. The control class will be processing and validating the input made by the athlete and the control class will the shoe which the data that will be persisted. These are now modelled in a collaboration diagram as shown below in fig 5.3.3b.

Fig 5.3.2b Collaboration Diagram



This process was repeated for a number of the other use cases, but not all of them. The AUP states that not need to be realised just enough to understand how the system will function. This process

helped consolidate what was needed in the data layer, business logic layer and the presentation layer of the 3-tier architecture.

5.4 Design

This part of the elaboration phase involves taking the requirements that have analysed through use case realisation taking into account functional and no functional requirements and designing the system around this **[20]**. This involves the design of the database, business logic involved and the user interface of the system.

5.4.1 Django and MVC

Django uses a Model View Control Architecture as stated in chapter 2. The Model will be visualised as the Entity, the View as the Boundary and the Control as the control. In Django terms the architecture is better described as Model, View, Template (MVT). The View is the Control and the View in MVC, is Template in Django **[11]**. The system will therefore be design around this idea. The section and the rest of the will discussed in terms of Model, View, Template.

5.4.2 Model Layer

In Django the model handles the database side of things. It provides a database-abstraction API that lets you retrieve, update and delete objects **[11]**. The database tables are represented as Python objects. So to create a database table you represent it as class in the model with the attributes you wish to store the object as. An instance of a class in the model represents a returned record. This means when interacting with the database Python code is written instead of SQL and is contained in the View. The use of database-abstraction means code written in the view is not tied to a data source and will not need to be re-written if this data source changes. With this in mind the entity classes derived in the previous chapter through use case realisation were mapped to objects that will be implemented in the model.

5.4.2.1 Database

The database design was based on the entity classes derived in the use case realisation. From this analysis, this will allow the tables in which the data will be stored to be designed.

Schema

As previously mentioned, from the entity classes the tables in the database were derived. The schema is as follows:

User (user_id, username, first_name, last_name, email, password, is_active)
Athlete (athlete_id, user_id, coach_id)
Coach(coach_id, user_id)
Event (event_id date, time, race_id, duration,, position, shoe_id, notes, user_id)
Race (race_id, name, type, distance, description,, user_id)
Route (route_id, name, distance, description, user_id)
Session (session_id, date, time, route_id, distance, duration, type, effort, shoe_id, notes, user_id)
Shoe (shoe_id, name, type, used, user_id)

A primary key was created for every table representing a persistent entity. A user_id foreign key was created on the Shoe, Race, Route, Session, and Event tables to allow the system to store it against the athlete. A user_id foreign key was used on the coach and athlete tables to allow a user to be a coach or athlete, the user_id of the user will be place in either the coach table or athlete depending on whether the user is a coach or athlete. The foreign key coach_id used on the Athlete table was done to associate a coach with an Athlete. These are all one-to-many associations.

Entity Relationship Diagram

An Entity Relationship diagram helps illustrate the design the database showing the relationships, high level abstractions and cardinalities between each relation. It also models the relationship between each entity in the database. The three types of relationship that can be modelled are: one-to-one (1:1), one-to-many (1:M), many-to-many(M:N).

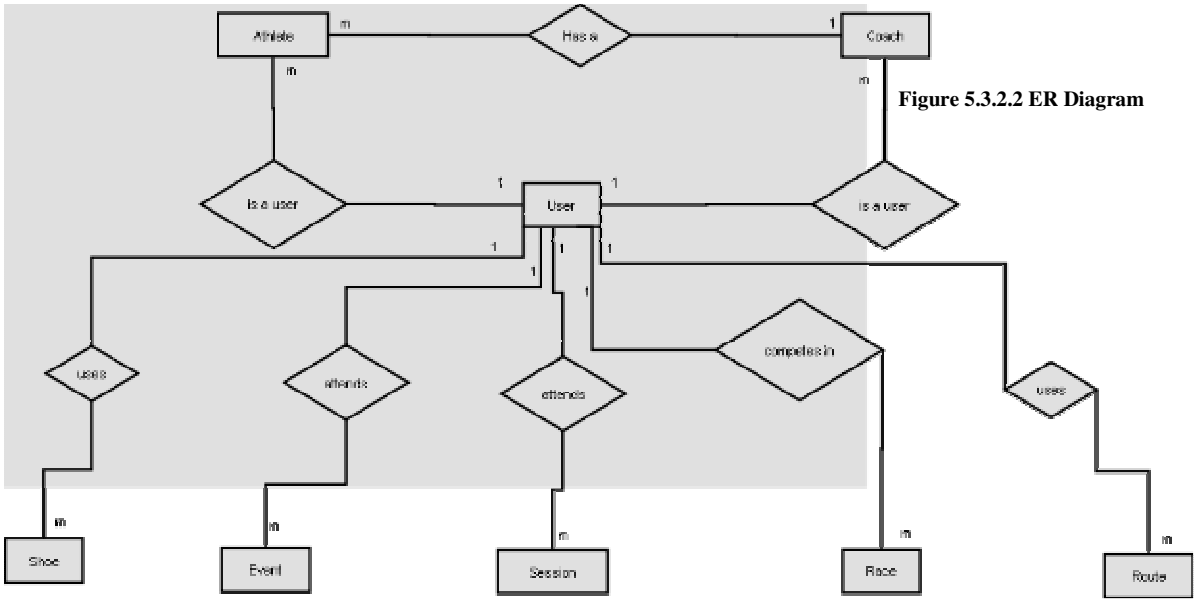


Figure 5.3.2.2 ER Diagram

The ER diagram shows 8 tables that represent how the database will be designed. The relationships are as follows: a user can has many Shoes, Events, Sessions, Places, and Routes. This is a 1:M relationship. The athlete and user table is a 1:M relationship because many athletes are a user but only one athlete is a user, the same is for the coach and user table. The athlete and coach table is 1:M because a coach can have many athletes but an athlete can only have one coach.

5.4.3 View Layer

This is where the business logic of the system is located in the MVT architecture. From the analysis performed, the structure of the business logic could be designed. The methods that relate shoes, routes, events, races, sessions, athletes and coaches were to be implemented in the views they relate. For example, the add shoe use case would be implemented in the shoes view, the view races method would be implemented in the races view. Designing it in this way will make implementing the functionality easier in the construction phase. So for athlete management, the use cases that represents management of shoe details such as adding a shoe, showing shoe usage etc, have been designed to be contained in the shoe view. In essence views for each boundary class will be created and the control classes relating to the boundary will be implemented in the view. The account management was designed in a similar fashion; with a view to contain the functionality need to handle user accounts created in a separate view. The run log, which will incorporate all the use cases bar the account management and the account management use cases, will be developed as two separate applications that will work together in unison to encompass the web application as a whole. Django allows and encourages the developer to split the application into separate "sub applications". In Django terms a web application is a Django Project and a sub application is a Django Application **[11]**. This will help when implementing the system.

The query and analysis tools were designed to be included as methods in the view they relate to. So the tools that calculate shoe usage and display a graphical output will be located in the shoe because that is the boundary that the control relates to.

The administrate features did not need to designed simply because Django comes with a built in administrator package.

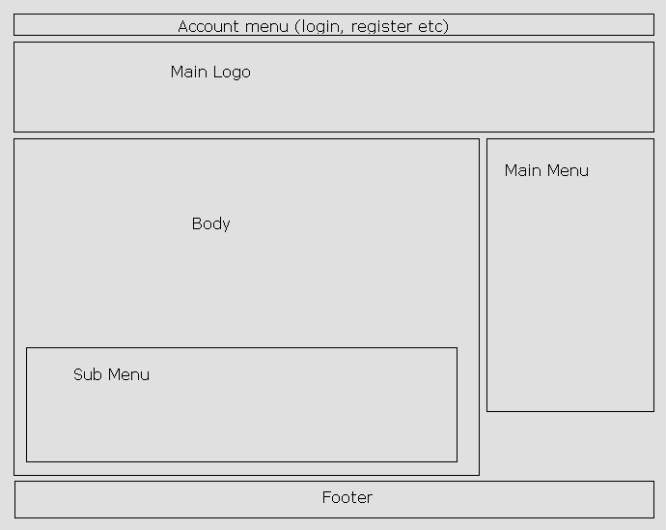
5.4.4 Template Layer

This part of the design involved designing the user interface of the system. Since this is a web application the user interface is a web front end viewed through a web browser. Web pages are made of standard elements such as a header, footer, main content, menus etc. An approach when designing a web application is to copy HTML in each user interface. Django discourages this approach by implementing a template system **[11]**. A base template is designed which contains all the elements that are required for the general user interface of the application. This avoids having to copy and paste into each user interface and any errors or bugs found need only be corrected once. The appropriate user interface relating to each view is then plugged in to this base template.

5.4.4.1 Template design

The original prototype had a basic user interface implemented. The design for the new user interface is a modified version of this template. Bearing in mind the usability checklist Larisa Thomason **[22]** set out, the template was designed as follows:

figure 5.4.4.1 User Interface Design



The design for the user interface is to be kept consistent to throughout, every web page that makes up the user interface will conform to this design. Larisa Thomason **[22]** states the importance of this as to not confuse the user by changing the user interface from screen to screen. There is clear navigation system in place also, allowing the user to easily navigate to the functionality of the web application they wish to use. The content section located in the body is clearly set out meaning the user will not have to spend time trying to find it. The brand of the application i.e. that is what the application does is set out in the title section of the template. The design of the user interface has incorporated usability factors to allow the application to be used to its full potential, because poor usability can put the user off from using the application **[2]**.

Chapter 6: Construction Phase

6.1 Introduction

In this phase of develop the goal is produce is to produce a working application which meets the requirements set out in chapter 2 including the functional and non functional requirements **[17]**. The implementation was done in iterative manner, with the account management and athlete management in the first iteration. In the second iteration, the Coach interface was implemented and finally the query tools were implemented in the final iteration.

6.2 Model Layer and Database

The model layer in the MVT was implemented first due the fact Django insists that the developer does this but also because the views that will be implemented after depend on having a model in place to store, retrieve and manipulate data. This is the data layer of the application. The model for the account management was developed and written in python code as was the model required for the run log part of the application. The models were based on the design of the database done in the previous chapter.

Fig 6.2 Model class for a shoe

```
class Shoe(models.Model):

    """Represents a pair of running shoes."""

    name = models.CharField(maxlength=80, unique_for_date='used')
    type = models.CharField(maxlength=8, choices=constants.SHOE_TYPES)
    used = models.DateField('date first used', null=True, blank=True)
    user = models.ForeignKey(User)

    class Meta:
        ordering = ['id']

    class Admin:
        pass
```

The fields for the database are variables in this class which represents the Shoe Table. After this all the necessary tables set out in the design were written as class in the model files and an empty database was created in SQLite. Django creates all the tables in the database for you based on the class you written in models.py file. To actually create the tables in the database a command is executed in the console:

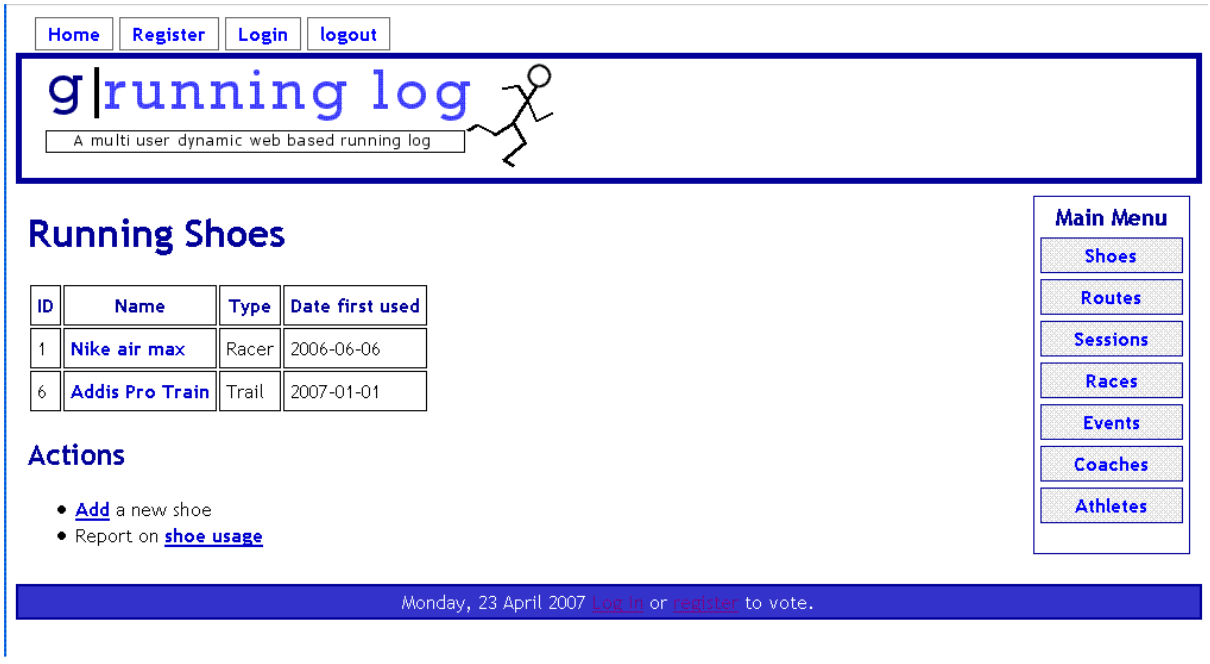
```
python manage.py syncdb
```

What this does is look at the models.py and cerates a table for every class and columns for every variable in that. This also checks the models file for any errors and reports them back. The first time the database is synced, the built in admin application is created for the application.

6.3 Template Layer and User Interface

The presentation layer of the application was implemented next. Following the layout outlined in the previous chapter and editing the current layout already in place for the existing prototype, the base template was created. The point of this is that gives the user, a user interface in which to interact with the application. This was done using HTML and CSS and formed the structure of the user interface. This base template created allowed the user interface to have consistency throughout the application. All the main links to the various functionality available were included in this template.

Figure 6.3 Screenshot of User Interface



Along with the base template, templates were created that would handle the user interface for the functionality of the system. These included forms to allow the athlete to submit data as well as pages that allowed the athlete and coach to view information, be that entered data in the system or some statistic such as shoe usage.

6.4 View Layer and Business Logic

After the user interface was developed and the database set up, the step is to implement the business logic of the application. This was done iteratively. A view was created to handle registration process but was not needed to handle the accounts because the built in admin application Django provides allows you to do this. This allows you mark users active in the system. Views were created also to handle each aspect of the run log, this included views for shoes, races, events, sessions, routes as well as well as coach and athlete views. The coach and athlete views will contain business logic to set a coach and allow the coach to select an athlete that to view that the athlete has set as their coach.

6.4.1 Single User to Multi User System

The first implementation interaction involved turning the system from a single user system to a multi-user system. This involved implementing the account management first. This was done by writing method called ‘create_inactive_user’ in the model and ‘register’ in the registration view. The register method takes the data entered in the register form and stores it in variables. These variables are then passed to the method ‘create_inactive_user’ which stores the data in the database. This method sets the account to inactive and using the admin application provided with Django the administrator then sets this active once they have review the account and decided to allow the user to register. Django automatically stores the password as a salted hash for security reasons. The registration system also set the user to be a coach or athlete depending on the choice made by the user and stored the user in either the athlete table or coach table

The next part was to implement the athlete management. All the use cases relating to relating to this were implemented. Those use cases in athlete management that related to a shoe were created as methods in the shoes.py view and the same with routes, races etc these methods were written so that along with saving data, showing data etc they acknowledged who was logged on and stored and retrieved data against the logged in user.

6.4.2 Coach Interface

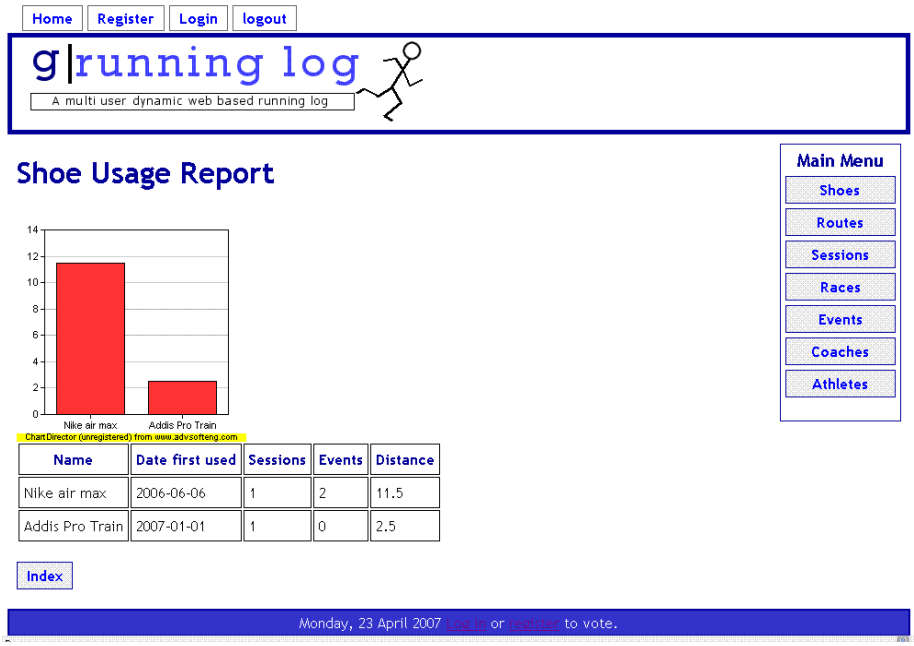
The next iteration implemented an interface to allow a coach to view their athlete's information. The two different users of the system had different needs. Each of the methods implemented previously were only implemented to handle an athlete. The coach interface was implemented by editing the methods above to give them the ability to check the type of user logged on. If a coach was logged on, they did not allow the coach to access any of the functionality such a view shoes, adding shoes etc unless they selected an athlete to view. Selecting and setting an athlete to monitor functionality was implemented in athletes.py view, which had a method, 'set_athlete,' that showed the coach the current athletes that have set them as a coach. From this list the method takes the id of the Athlete and stores them as a session. This session is then called when coach wants to view information about an Athlete and shows them information about their chosen athlete. Coaches were prevented from being able to add or edit anything about the athlete only able to view information.

The method to select a coach was implemented in similar fashion. A method called 'set_coach' produced a list of currently registered coaches and allowed you to set a coach. Only an athlete was able to select to coach, the coach was not able to add an athlete to view.

6.4.3 Analysis Tools

The next iteration implemented the analysis tools. This involved writing methods that took data and performed some statistical analysis. These tools were implemented to be made available to both coach and athlete.

Figure 6.4.3a Shoe Usage report with graph output



Data entered by the athlete was used to derive reports, such as the shoe usage report. A method 'usage' is used which finds all the shoes the athlete has stored. From this information retrieved by this bit of functionality a graph is also generated. In order to achieve this, a 3rd party library was used called ChartDirector **[23]** was used. In order to develop similar graph drawing abilities would have and taken too long and deemed pointless since there are a number or libraries available that do this. This library offers a number of methods to enable drawing graphs. A method called 'generate_graph' was implemented, which took data and produced a graph as image file which was included in the template for this part of the user interface. This method was called when the 'usage' method was called.

Another tool implemented was a search function that allowed the user to search sessions based on route and within a time period.

Various similar tools were implemented through the system to aid analysis of the athlete and allow the coach to view progress.

Figure 6.4.3b Search Sessions by date

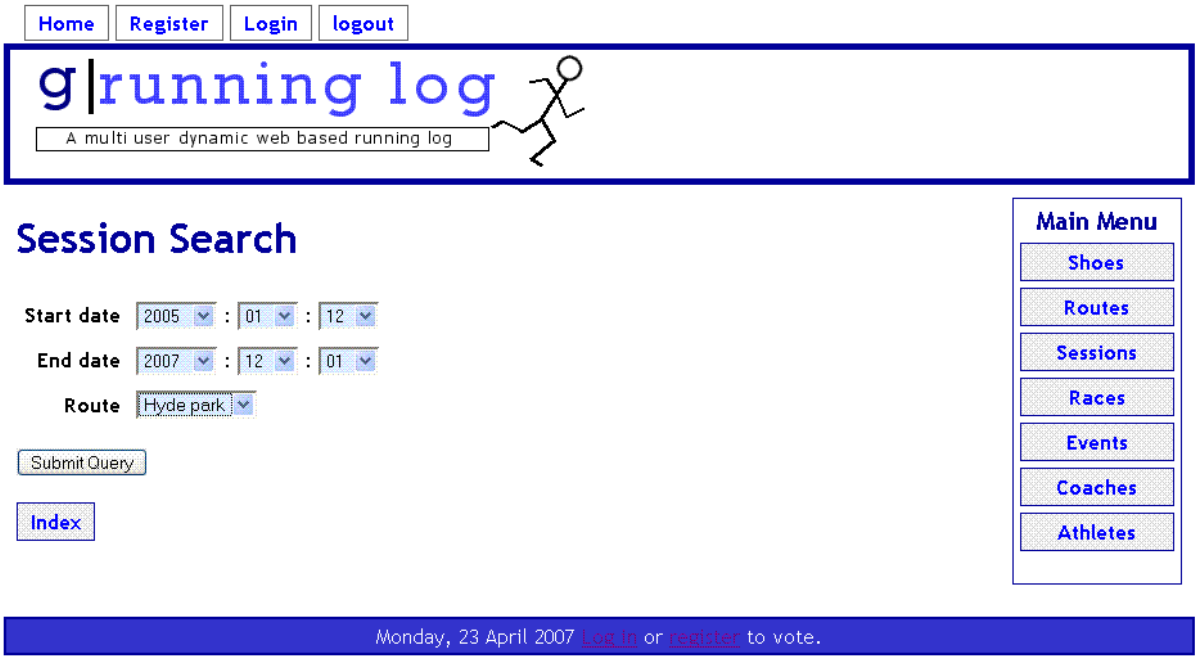
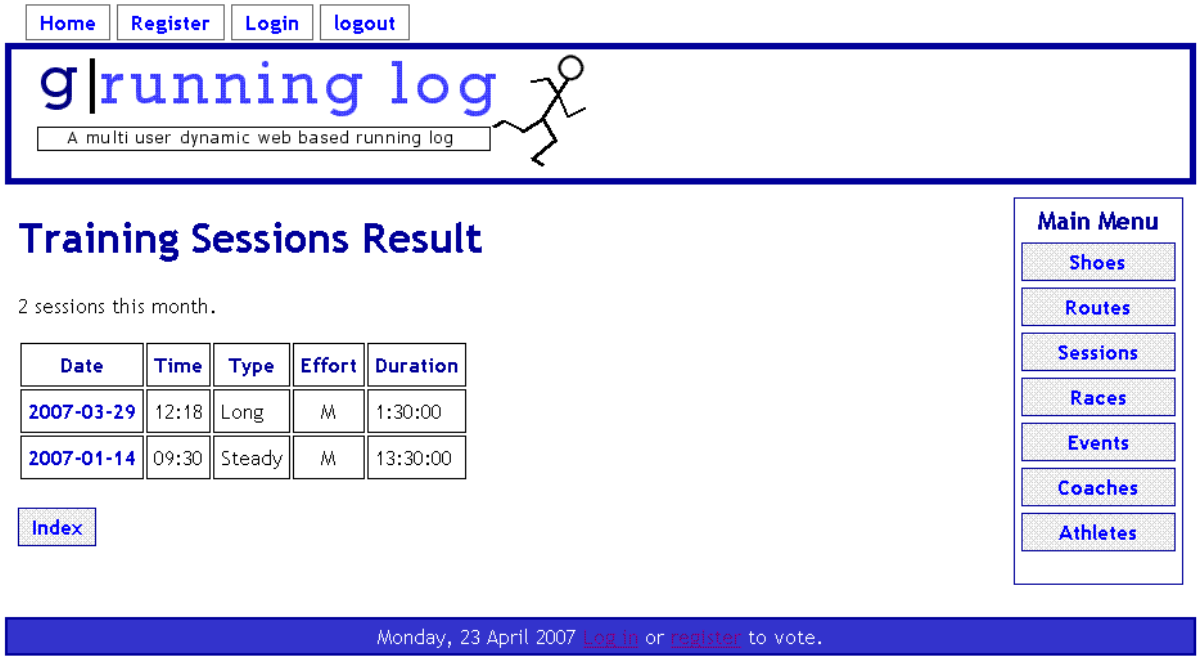


Figure 6.4.3c Results of search



6.4.4 RSS Feeds

The next implementation was a further enhancement. The ability for the system to add a RSS (Really Simple Syndication) feed and allow an athlete to view this feed with a link to the main article. Only the Athlete was given the ability to do this. To do this a method called 'add_rss' was added. This took the URL the athlete submitted and stores it in the database. Next a method that rendered the RSS was implemented. Since RSS is basically XML, this would have to be parsed first. A 3rd party module called 'parsefeeder' was used [24]. This module has methods that parse syndicated feeds like RSS and allows display the feed in anyway wished. The RSS feed stored in the database is fed into this module and the information is displayed. The view RSS implemented then takes this parsed RSS feed and displays it for the user if they have an RSS feed stored.

See Appendix D for screen shots.

Chapter 7: Testing

7.1 Introduction

Testing is important because it allows any defects, bugs and errors to be identified and gives feedback to the developer. In this case the application was tested after each new implementation was done to check the functionality worked and also to the check the basic and alternative flow of the system. This section discusses the testing that was done

7.2 Initial Testing

Testing was carried out after each use case group was completed. So when the account management and athlete management was implemented tests of the functionality were carried out and this was repeated as more functionality was added to the system. This meant that any bugs and errors were spotted and immediately corrected. Tests that were carried out include checking that data was stored accurately and in the correct place, that the different types of user, coach and athlete, were only able to do what they were supposed to do such as the coach not being able to add to or edit an athlete details, that they were only able to view athletes that had selected them as a coach. The query tools were test as well to ensure they deliver accurate results. Some of this testing involved manually checking the database to ensure details were correctly added to the system and this proved to be quite time consuming.

7.3 User Testing

After functionality was implemented it was necessary to test the application on users. The reason for this was two fold. The first reason was the test the usability of the system, the second was too check for any bugs or errors that may have been missed earlier.

The system was tested on two users. The users asked to use the system given a brief introduction and overview on how the system works. One user played the role of an athlete while the other played the role of a coach. The user playing the role of the coach was asked to register and was able to do this informed that the admin would activate their account soon, it was commented this was useful. Next the user was given a pre register account with some information filled and asked to perform tasks.

The user was not given a tutorial on the navigation and was asked to find things themselves. One comment from this was that the functionality of the system they wanted to use was easy and quick to find. The user commentated that the layout seemed logical and the menu system easy to use. Overall the user commented that the navigation system was excellent and when asked if they ever felt like they were lost, the answer was 'no'. The user was asked to comment on the aesthetics of the system such as the colour scheme and readability of the text. It was noted that the colour scheme worked well apart from a one link at the bottom that was hard to read. This was a link in the footer telling the user if they were logged in. Some minor bugs were brought to attention. The search function on the search session interface had to be fully entered to for the search to work and the user commented that it would be better if the search allowed only partial entry.

The second user was asked to run through the coach interface, similar comments about the usability were made along with the search function above.

For a summary of User Testing see Appendix

7.4 User Interface

Along with user testing, the user interface was matched against the usability guidelines set out by Larisa Thomson **[22]**. The User interface met most of the guidelines set out, such as a clear navigation system, clear content, and supporting the brand of the system and consistent look.

It was felt the usability of the system as a whole was good and with relatively little guidance the system could be used effectively.

Chapter 8: Evaluation

8.1 Introduction

This evolves evaluating the project to determine the success of it. A number of areas will be evaluated in this section including: Time Management, the methodology used and meeting minimum requirements. The system will evaluated against: correctness, robustness, usability, performance and security. Finally any future improvements to the system will be discussed.

8.2 Methodology

The methodology that was followed was the Agile Unified Process (AUP) that was modified for the purpose of this project. The main modification was to adapt it from team based software methodology for use with one person. Also necessary was adapting it to fit how Django works because Django does not work in a purely object orientated manner.

Using this methodology helped a lot with the project, in enabled clear stages and goals to be set and allowed the project to split up into more manageable segments rather than tackling as one mammoth task. It also helped in understanding the problem at hand as it was approached from a high level and as each new phase was entered, a lower level view of the project was seen. This helped immensely when it came to deciding how implement functionality.

Another advantage of using this methodology was the agile nature of it. It did not require stages to be completed in immense detail. Analysis and design were done enough to allow the system to be implemented. This was useful because it meant that becoming too engrossed in certain parts of the project and not moving on when it was necessary didn't happen and things progressed as quickly as possible.

8.3 Schedule

The planned schedule and revised schedule can be seen in Appendix B. Due to implementation issues the schedule had to be revised some what to factor in the longer time needed to actually program. This was due to the fact the initial time given to programming the system was not enough. It was originally planned to have most of the system implemented by the beginning of march, but in fact was

not completed until the second week of the Easter holidays. After this set back was factored in, the rest of the schedule was kept to. Before the implementation stage the schedule was adhered too. The original schedule was also not detailed enough in outlining the work and this was revised in January after the exams to factor in all the phases of the AUP.

8.4 Minimum requirements

All the minimum requirements for this project were met:

- *Further develop current prototype to support multiple users.*
- *Develop User Interface to support coaches.*
- *Create tools for Query and Analysis of training data.*

How these were implemented in the system in outlined in chapter 6. In order to meet these requirements the methodology was followed. The database and application were implemented according to the analysis and design carried out. Overall the system met the goals and requirements set out and exceeded the minimum requirements by implementing the *'Add RSS feed to log'* further extension.

8.5 Technologies

Having had little experience with web technologies before starting this project there was some what of learning curve. The Django Web Application Framework is Python based. Having had previous experience of Python in past university modules such as *SE10: Fundamentals of Programming* and *SY23: Internet Systems Technologies,* this meant a whole new programming language did not have to be learnt, but figuring out how Django worked proved more challenging. One of problems was that some of the documentation was incomplete and the book was not finished. This meant when it came to implementing certain functionality it took a lot longer than expected. An example of this would how Django handles forms. There is very little documentation on how to use forms within Django, so this took a lot of searching on the World Wide Web, asking questions in the Django IRC channel and viewing sample implementations. This is mainly due to the fact that Django is a relatively new Framework having only been released in the summer of 2005. It is still heavily in development and current the version number is 0.96.

Getting to grips with how to develop a web application in Django also took some time to learn. It uses a Model View Control style framework, or as Django calls it Model View Template. Having had

not much experience of developing in this manner, again this took time. One of the good things about Django was it's use of a database-abstraction API. This meant that no SQL was needed to be written to access data in the database, instead python code was written. This was nice feature because it made accessing and manipulating data a lot easier. It also took some of the tedium that can happen when writing SQL.

Another positive is the great community support Django has. The Django website provides a mailing list and links to many community sites. Also the Django IRC was extremely useful, offering great advice and a lot of technical knowledge to supplement learning.

8.6 System Evaluation

The following section evaluates the system based on the criteria mentioned in the introduction of this chapter. This was done with the aid of the user testing that was done.

8.6.1 Correctness

Are training and racing details stored and retrieved reliably?

The training and details are stored accurately by the system. In order to evaluate this, the data stored by the system was manually checked in the database. For example, a shoe was stored by an athlete, one the system had added the shoe, the database was checked manually to see if what was entered was actually stored in the same in the database. This was done for all methods that added to the database. The check retrieval, a similar process was employed. When checking a user's list of shoes, it was noted who the user was and the details of the shoe. The database was manually checked to ensure that the shoes being listed are the ones that are meant to be listed.

Do queries against the database return accurate results?

This was evaluated again by manually checking the database manually against the queries made in a similar fashion to above. When all the queries were checked they all returned accurate results.

Are statistics or other forms of analysis computed accurately?

To evaluate this the statistics and analysis computed by the system were noted. After this the results they were produced were checked by manually by calculating them with pen and paper. This was deemed the most accurate way of evaluating accuracy. The results computed matched those on paper

8.6.2 Robustness

Can the system cope sensibly with attempts to enter bad data?

The system does cope with bad data entered. Bad data might be something like typing a string instead of date. The system validates any data that is entered, it checks the data, if the data is bad, then the system does not add it to the database and requests the user re-enter the correct form of data instead.

Is data corruption avoided when a component of the system crashes?

Data corruption is avoided by making use of transactions. It works by this, when a request is made, Django starts a transactions and commits if there are no problems. If exception is made or the system crashes, Django automatically rolls back any transactions **[11]**. This is a built in feature in Django to help avoid data corruption.

8.6.3 Usability

Can use cases be enacted without difficulty?

When the users were testing the system, they were given instructions to perform task that related to the use cases. From the testing the users had no problems performing the task and they did not ask questions to try and find correct part of the system that related to the use case. In the system the use cases can be enacted easily and without error

Is the UI sufficiently easy to understand and use?

The User interface used in this system followed an easy to use structure. There were 7 main links relating to different aspects of the training log such as shoe, races etc. Whenever one of these links is chosen, a sub menu giving links to the various options available is presented. The users who tested the system commented that this easy to understand and had no problems navigating around the User Interface.

Does the UI function correctly on different platforms?

Since the User interface is a web based one, it was checked that it ran on a number of browsers a platforms. The User interface functioned correctly on MS Windows using Internet Explorer 7, Firefox

2.0 and Opera 8.5. On Linux the user interface functioned on Firefox 2.0 and Opera. There were no problems in running the system on different systems

Is the UI sufficiently accessible to those with disabilities?

The User Interface used a minimal amount of images and only when deemed necessary where they used. The main disability factors to take into account are those with poor sight and those who are blind. Since the system is predominantly text based, it will allow screen readers to use it effectively. Also for those with poor site, the text size can be changed by using the standard text size functionality in web browsers.

8.6.4 Security

Are secure passwords for athletes and coaches enforced?

Passwords stored by the user can be anything they want. This could lead to users storing unsafe passwords by the user. Forcing a user to use a password they don't want to use, although more secure, could put them off using the system and also could lead to user forgetting their password. The decision was taken not force the user into having certain type of password. The password they choose is stored securely in the system. It is stored as a salted hash, which scrambles the password., which means that if anyone did compromise the security of the application they would not get access to user's account password because the salted hash is unreadable. Even the site administrator cannot view user's passwords as in the admin interface the password is shown as a salted hash.

Are permissions set appropriately for athletes and coaches?

Whenever a user registers to use the system, they must select to be either an athlete or a coach. In the system coach's only have access to athlete's information if an athlete has set them as a coach. The access they do have is only to view the information about an athlete, they are unable set to edit or add information about their athlete. The permission set up also mean an athlete can only access their own information, they cannot view, add or edit any other athelete's information that is stored

Are administrative functions protected adequately?

When the system is setup initially, Django advises you create a 'superuser'. This 'superuser' is the administrator of the system. Only this user has access to the admin backend of the system. If other

users try to access the admin backend they are denied access. The administrator can allow other users to access this section if desired but by default only the initial 'superuser' has access to admin functions.

8.7 Improvements

There are many improvements that could be made to the system. Of the additional requirements, a web forum could be implemented allow users to training issues. The user interface could be further developed to incorporate some Ajax techniques. Also the use of a mapping tool to allow users to visually plot the routes and races they use would prove most useful. An internal messaging system may be another area that could be developed to allow coach and athletes to communicate with each other and possible allow some sort of blogging tool could be incorporated to allow an athlete to make notes.

8.8 Conclusion

Overall the project was a success, the minimum requirements were met and exceeded and system had the desired functionality implemented.

References

[1] Florence Glover, S. 1999. Reruns—the training diary - keeping a running diary can prevent problems - Brief Article. [accessed 19th October 2006]
Available from World Wide Web:
http://www.findarticles.com/p/articles/mi_m0NHF/is_3_17/ai_86649606

[2] Jakob, N. 2003. Usability 101: Introduction to Usability [accessed 17th March 2007]
Available from World Wide Web:
http://www.useit.com/alertbox/20030825.html

[3] Serpentine Running Club. Running Logs for beginners [accessed 4th December 2006]
Available from World Wide Web:
http://www.serpentine.org.uk/advice/begin/logs.php

[4] Running-Log.com [accessed 5th December 2006]
Available from World Wide Web:
http://www.running-log.com

[5] RunningAHEAD [accessed 5th December 2006]
Available from World Wide Web:
http://www.runningahead.com

[6] McPherson, S. 2000. JavaServer Pages: A Developer's Perspective. [accessed 6th December 2006]
Available from World Wide Web:
http://java.sun.com/developer/technicalArticles/Programming/jsp/

[7] Darie, C & Ruvalcaba, Z. 2006. Build Your Own ASP.NET 2.0 Web Site Using C# & VB. pp 2-5 SitePoint Pty. Ltd.

[8] Hart C, Kauffman J, Sussman D, Ullman C. 2006. Beginning ASP.NET 2.0 with C#. pp 7-9. Wiley Publishing, Inc..

[9] Bogab D. 2006.Get On track with Ruby on Rails [accessed December 5th 2006]
Available from World Wide Web:
http://www.sitepoint.com/article/ruby-on-rails

[10] Holovaty A, Kaplin-Moss J. 2006. Django Book [accessed December 5th]
Available from World Wide Web:
http://www.djangobook.com/

[11]Django, 2007, Django Documentation [accessed March 18th 2007]
Available from World Wide Web:
http://www.djangoproject.com/documentation/

[12] PostgreSQL, 2006. About [accessed 7th December]
Available from the World Wide Web:
http://www.postgresql.org/about/

[13] MySQL, 2006. What is MySQL? [accessed 7th December]
Available from the World Wide Web:
http://dev.mysql.com/doc/refman/5.0/en/what-is-mysql.html

[14] SQLite, 2006. About SQLite [accessed 7th December]
Available from the World Wide Web:
http://www.sqlite.org

[15] Bennett S, McRobb S, Farmer R. 2002. Object-Orientated Systems Analysis and Design. pp 57, 135-144, 161-167.
McGraw-Hill.

[16] Blue Ink, 2007. Rapid Application Development [accessed 25th March 2007]
Available from World Wide Web:
http://www.blueink.biz/RapidApplicationDevelopment.aspx

[17] Ambler S W, 2006. Agile Unified Process [accessed 8th December]
Available from the World Wide Web:
http://www.ambysoft.com/unifiedprocess/agileUP.html

[18] WISDM, 2007. WISDM [accessed March 27th 2007]
Available from the World Wide Web:
http://www.wisdm.net/wisdm/index.htm

[19] Vidgen R, Avison D, Wood B, Wood-Harper T. 2002. Developing Web Information Systems pp 31-33
Butterwoth-Heinemann

[20] Avison D, Fitzgerald G. 2003. Information Systems Development: Methodologies, Techniques and Tools 3rd Edition. pp 89-94, 97, 428

[21] Efford N, Johnson O, 2006. SE20: Object-Orientated Software Engineering 2005 -2006.
The University of Leeds

[22] Larisa Thomason, 2004. Website Usability Checklist.[accessed February 21st 2007]
Available from World Wide Web:
http://www.netmechanic.com/news/vol7/design_no4.htm

[23] Advanced Software Engineering(ASE), 2007 ChartDirector [accessed 2nd April 2007]
Available from World Wide Web:
http://www.advsofteng.com/

[24] Mark Pilgrim , 2006 Feed Parser [accessed 18th April 2007]
Available from World Wide Web:
http://www.advsofteng.com/

Appendix A: Personal Reflection

This Final Project was an interesting and enjoyable experience although at times proved quite stressful. The project has enabled me to gain new skills not covered over the course of my degree and enhance those that I already have. It also allowed me to broaden my knowledge of web applications and how they work and in particular the Django Web Application Framework.

Django is good technology to use. I would recommend that if future students attempt a similar project that they spent a significant amount of time getting to know the technology they use. The time it took to acquire the necessary knowledge required to be able to actually do the project was a lot more than expected.

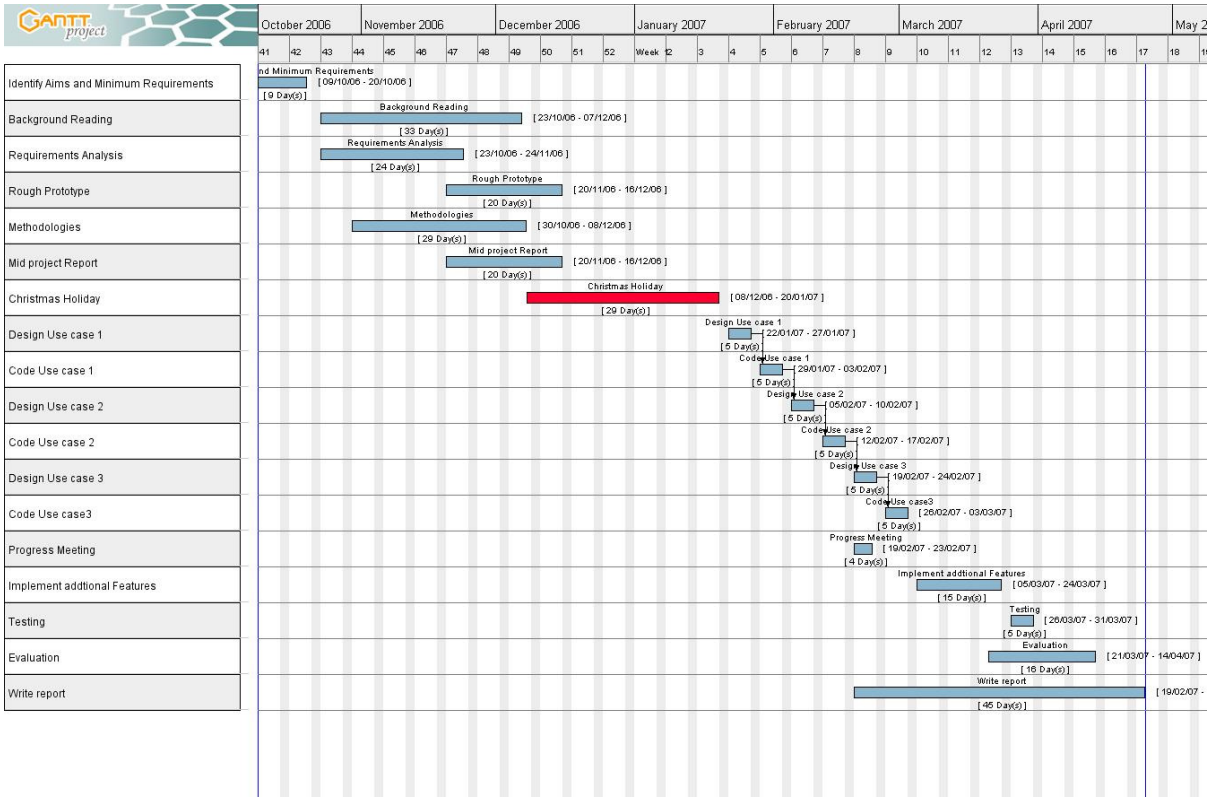
Setting aside time in the project to do this should be considered by future students. Students may also wish to consider using a technology that provides a significant amount of documentation to support. One of the main problems I found in this project was the incomplete documentation provided. Many of the manuals and tutorials had parts missing, in particular the documentation on how to use forms within Django was more or less non existent. Future students need to research carefully what technology they are going to use.

When it came to writing the report, there were many challenges. I have never written a report of this magnitude in my university career. The sheer size of it alone was daunting. Also explaining the use of technologies used within the project without being too technical was difficult. As advice for future students I would recommend that you start writing the report as early as possible. I feel started mine too late and this put me under a lot of pressure. Allocate about twice the amount of time to writing the report. I would also advise future students to sort out the structure of the report before writing the content, this will make writing the report a lot easier because you will have an idea of what you will be writing about and overall the report will seem less daunting.

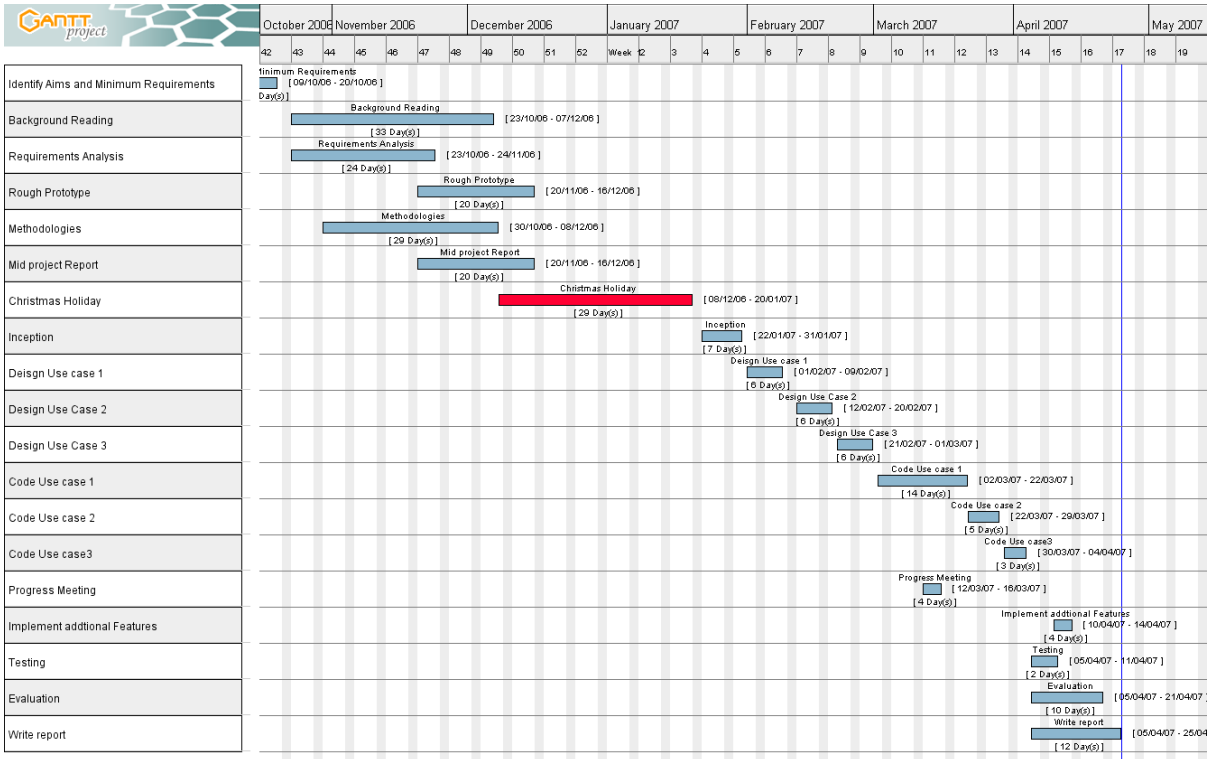
Overall I feel the project was success. Having had little experience with web development other than developing HTML static pages I was able to fully implement a working web application. The main bit of advice I would give to future students is to choose a project you will enjoy doing. You will be spending a lot of time doing this, so make sure it is something you look forward to doing rather than looking at it as a chore. This will help a lot with doing the project.

Appendix B: Gnatt Charts

Initial Schedule



Final Schedule



Appendix C: Use Case Description Form Example

*	Use Case Name: *(The name as it appears in the Use Case Model)*	**Add new shoe**
*	Primary Actor: *(Actor that initiates Use Case)*	Athlete
R	Other (Secondary) Actors:	
R	Value Proposal to Actor(s) *(the goal of the Use Case from the Actor's perspective)*	To store the details of an athlete's shoe in the database for the current athlete using the shoe
R	Basic Course of Events: *(The Normal Flow)*	This use case begins when the athlete has a new shoes to store. The athlete navigates to desired form to add a shoe. The athlete stores details of the shoe which include, name of shoe, type of shoe and when first used. The athlete performs a request to save details of the shoe The shoe details are validated The system stores the shoe details in the database The system displays a list of the athlete's stored shoes
	Alternative Paths: *(Other paths through the use case which result in a successful outcome – typically variations to the basic course of events, determined by the actor and their needs).*	If the athlete enters incorrect information they told what it is and allow to resubmit the correct shoe details
	Exception Paths: *(Other paths through the use case which result in an unsuccessful outcome – typically when something goes wrong)*	If the user cancels and returns to homepage
	Assumptions:	
	Pre-conditions:	The systems knows who the user is

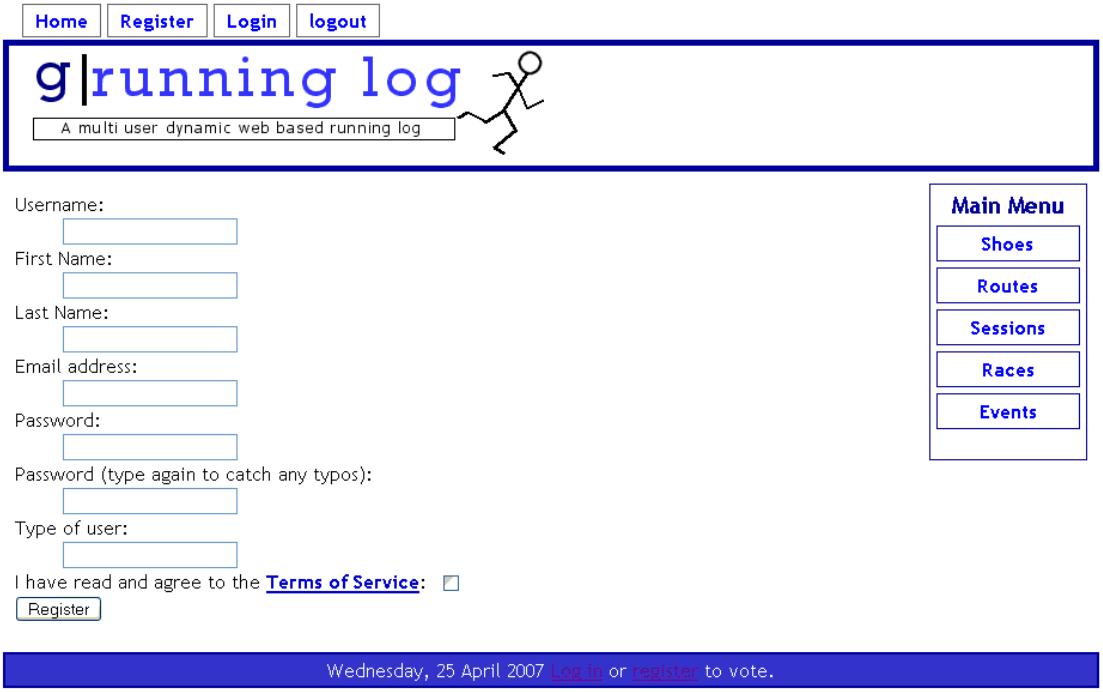
	Post-conditions:	The system displays a list of the shoes stored in database for athlete
	Related Business Rules: *(Reference to your Business Rules list)*	
	Related Non-Functional requirements – Usability, Performance, Security: *(Any non-functional requirements that are specific to this Use Case rather than the system as a whole)*	Ensure the data is in valid format
*	Project:	Running Log
*	Author:	Gareth McAteer
*	Date:	01/02/07

Appendix D: Screen Shots

Here are few more screen shots of the system in action:

Login Screen

Register Screen



View RSS Feed Screen

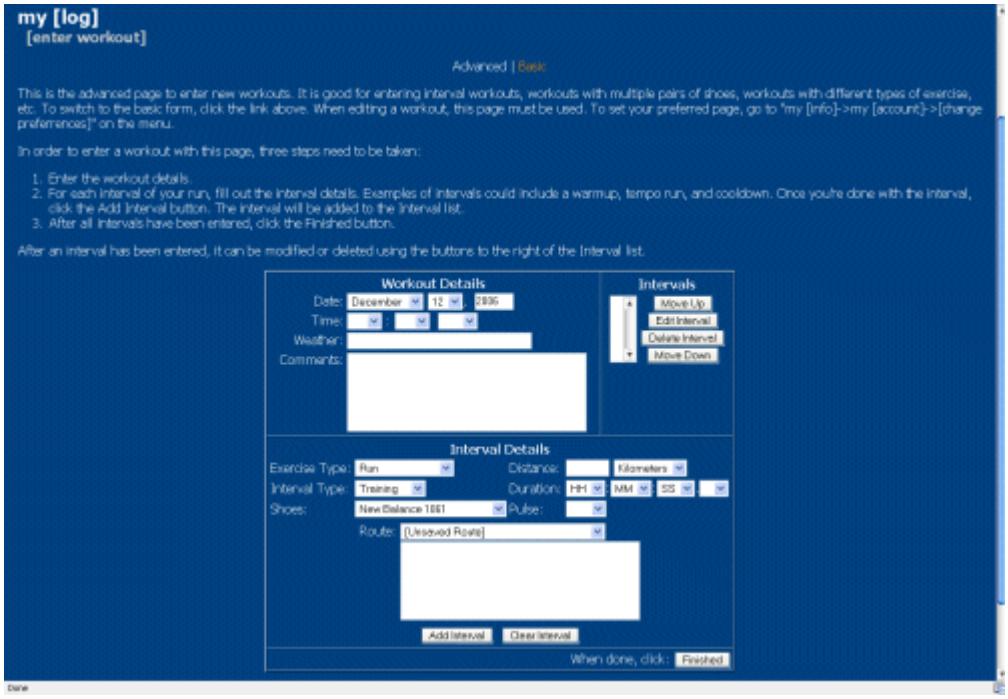
Add RSS Feed

Set Coach Screen

Select Athlete screen

Appendix E: Screen Shots of alternative systems

Running-Log.com


my [log]
[enter workout]
Advanced | Basic
This is the advanced page to enter new workouts. It is good for entering interval workouts, workouts with multiple pairs of shoes, workouts with different types of exercise, etc. To switch to the basic form, click the link above. When editing a workout, this page must be used. To set your preferred page, go to "my [info]->my [account]->[change preferences]" on the menu.
In order to enter a workout with this page, three steps need to be taken:
1. Enter the workout details.
2. For each interval of your run, fill out the interval details. Examples of intervals could include a warmup, tempo run, and cooldown. Once you're done with the interval, click the Add Interval button. The interval will be added to the Interval list.
3. After all intervals have been entered, click the Finished button.
After an interval has been entered, it can be modified or deleted using the buttons to the right of the Interval list.
Workout Details
Date:
Time:
Weather:
Comments:
Intervals
Move Up
Edit Interval
Delete Interval
Move Down
Interval Details
Exercise Type:
Interval Type:
Shoes:
Distance:
Duration:
Pulse:
Route:
Add Interval
Clear Interval
When done, click:
Finished

RunningAHEAD